VIZN: Illuminating Faces

Jacob Mccalip[#1], Daemon Mutka[#2], Julio Cantu[#3]

Texas State University

[1]jsm246@txstate.edu

[2]daemon.mutka@txstate.edu

[3]jar710@txstate.edu

***Keywords*— AdaBoost, Skin Detection, Cascade Classifiers, Facial Recognition.**

I. Introduction

Our final machine vision project, VIZN, was made with the intention to collectively use what we learned in class to detect faces from different images. To do so we used Adaboost as the backbone and Cascade Classifiers as its framework to build the trained model. For detecting we also implemented a Skin Detection functionality to heavily reduce the data the model will need to sort through.

II. Methodology

We decided to use AdaBoost paired with Skin Detection and Cascade Classification, respectively. We wanted to get an AdaBoosted trained model to first be able to detect faces and wanted to make the process more accurate by adding skin detection and Cascade classification. The Skin detection takes in the given data and outputs a mask, which is further refined using Cascade classifiers to break the detection to stages that start fast and inaccurate, but further stages take longer to compute but are far more accurate.

A. AdaBoost

AdaBoost is a machine learning algorithm that is mainly used for classification, usually in the form of combining multiple weak and fast classifiers into stronger, more complex ones. AdaBoosts ability to combine weak classifiers and ability to comprehend extreme examples makes it a great tool to use to train our program for facial recognition using a strong classifier built from weaker smaller classifiers.

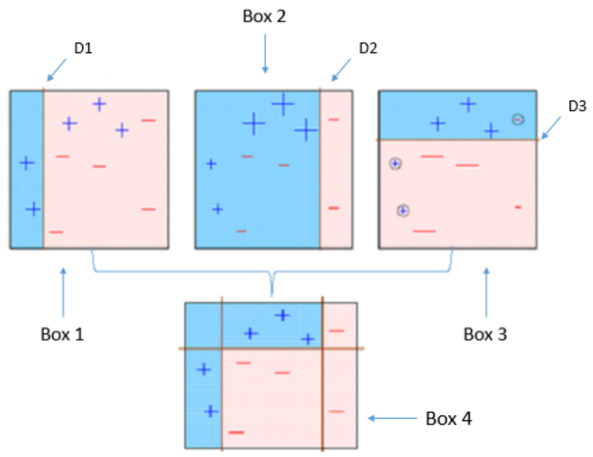


Fig. 1.1 An illustration of AdaBoost shows D1, D2, and D3 as regions derived from weaker examples. These regions are amalgamated in Box 4, resulting in a stronger and more precise method of detection.

B. Skin Detection

We decided to add skin detection to our project to better solidify the results from Adaboost and increase the accuracy of facial recognition. The basic skin detection functions by making a mask that will show the Adaboost only locations where skin is, from there Adaboost can make a more accurate classifier with less negative results.

C. Cascade Classifiers

Cascade classifiers involve a sequence of classifiers, starting with a rapid but generally less precise initial stage, followed by subsequent stages that are increasingly slower but more accurate. That is, the early stages of the process are designed to quickly eliminate regions that are unlikely to contain a face, whereas the later stages focus on refining and accurately identifying the regions that are more likely to represent faces.

This procedure of face detection leads to the process of facial recognition being faster between each image as the classifiers quickly discard regions that don't have facial features that the skin detection may have added.

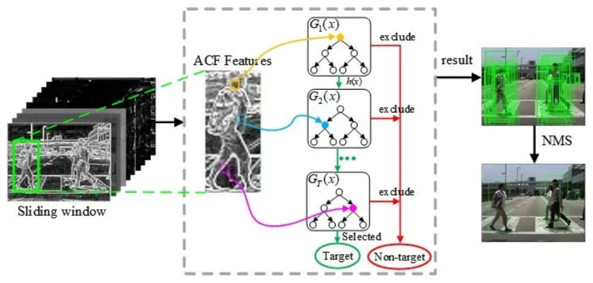


Fig. 2.1 Overview of Cascades Structures.

III. Data

The data used to train AdaBoost is mainly black and white bmp files containing faces, along with jpg files that are not composed of any facial features. Basic skin detection is trained and used with a UC Irvine provided histogram, while advanced skin detection uses an assortment of photos to create the skin detection mask.

A. *Data type*

The type of data used for the AdaBoost training functionality can be any image. The files used for training for facial recognition are located inside the folder labeled training_faces and are composed of grayscale bmp files cropped that each show a single face. However, the files in the training_nonfaces subfolder are .jpg files that feature a diverse set of objects from landscapes, to animals, and even everyday items that should not classify as a face.

The type of data we used for the basic skin detection addition was a skin/non skin histogram from UC Irvine, "*The Skin Segmentation dataset is constructed over B, G, R color space. Skin and Nonskin dataset is generated using skin textures from face images of diversity of age, gender, and race.*"[1] Using the histogram is used to compute mean and standard deviation for the red and green color channels.

Unlike the basic version, the Advanced Skin detection is implemented based on the correlation rules between the Y*C*b and Y*C*r subspaces and takes into account dynamically defined skin cluster ranges. The method was introduced by Brancati et al. which incorporated a methodology that is able to take into account different lighting conditions [2]. "The shape and the size of the clusters mainly depend on the lighting conditions and are identified on the basis of the information of luminance values Y in the current image. This approach makes our method robust to changes in lighting conditions, because it works on non-predefined clusters that are identified as image-specific skin color sets."[2] Using this methodology, we were able to create a more advanced skin detection method that is able to accurately detect skin ranges in different lighting variations.

B. *data explanation*

Below are the 3 different methodologies along with the runtime, correct amount, incorrect amount, and the accuracy of each.

TABLE I

Methodologies combined and corresponding data

Methods, adding going down	**Data**		
	Training time	**Runtime**	**Averaged Precision between 3 test folders 100% = 1.00**
Adaboost	17.96 sec	124.06 sec	.830
With Skin Detection	17.95 sec	125.83 sec	.853
With 12 stage Cascade	600.31 sec	47.84 sec	.876

As shown above, adding skin detection added minimal time to the runtime while increasing the precision by .02 (2%). On the other hand, when adding the 12 stage cascade it greatly increases the training time while decreasing the run time by a little more than half.

IV. Design and architecture

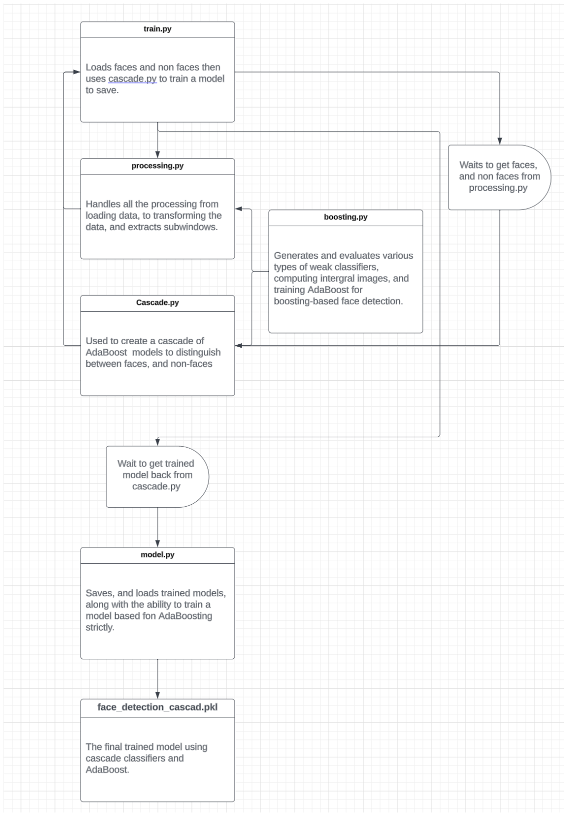
Our project is structured with distinct directories, with a focus on facial detection. At the top level, the two fundamental functions, "train" and "test". provided are two flow charts that visualize how the code functions and operates with one another.

A. *Overview of projects design*

We organized the project to provide clear and straightforward navigation, especially for those unfamiliar with our tech stack. The main directory includes two primary functions: one for training and another for testing, each located in their respective subdirectories. Each subdirectory is clearly labeled to reflect its contents. For example, the 'src' subdirectory houses all our source code, whereas the 'data' subdirectory comprises all the raw data used for training and testing our models.

B. *Explanations of interactions*

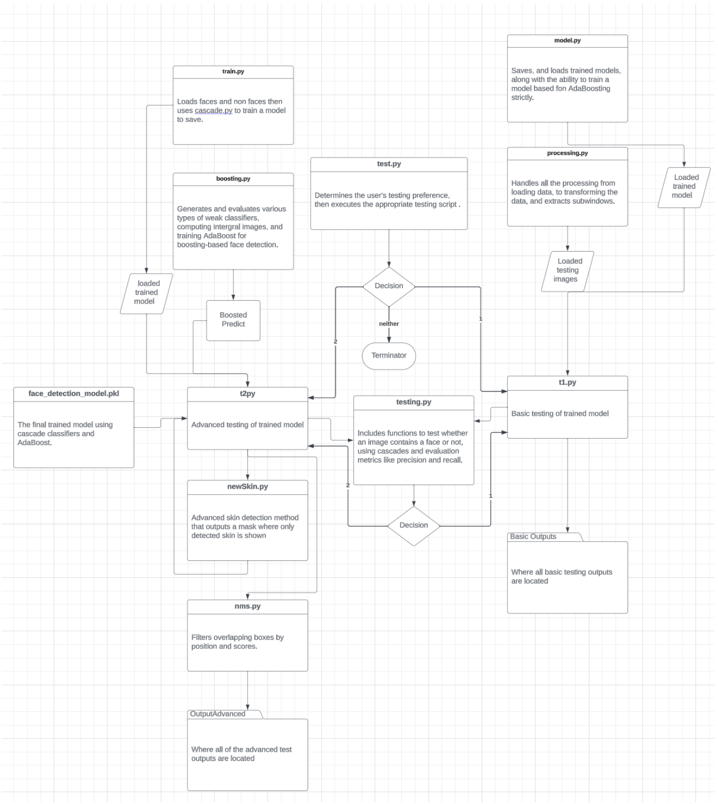
Below is a flowchart of our source code and interactions between each other, the left side is the training method, while the right side is the two different training methods.



train.py
Loads faces and non faces then uses cascade.py to train a model to save.
processing.py
Handles all the processing from loading data, to transforming the data, and extracts subwindows.
Waits to get faces, and non faces from processing.py
boosting.py
Generates and evaluates various types of weak classifiers, computing intergral images, and training AdaBoost for boosting-based face detection.
Cascade.py
Used to create a cascade of AdaBoost models to distinguish between faces, and non-faces
Wait to get trained model back from cascade.py
model.py
Saves, and loads trained models, along with the ability to train a model based fon AdaBoosting strictly.
face_detection_cascad.pkl
The final trained model using cascade classifiers and AdaBoost.

Fig. 3.1 Flow chart of training functionality

To start with the most important area of our code, the training functionality, it starts off by calling train.py. Following the command call it pulls data for faces and non faces, from their respective locations in the directory, and starts our train cascade function with the datasets pulled. Once the function is called, a cascade of AdaBoosted models start to form, increasing the amount of classifiers at each stage ending at 12. Once the training in model.py is complete, the "save_model" function is invoked from train.py. This function takes the fully trained cascade model and saves it in our directory, specifically within the "training" folder.



model.py
Saves, and loads trained models, along with the ability to train a model based fon AdaBoosting strictly.
train.py
Loads faces and non faces then uses cascade.py to train a model to save.
test.py
Determines the user's testing preference, then executes the appropriate testing script .
processing.py
Handles all the processing from loading data, to transforming the data, and extracts subwindows.
Loaded trained model
boosting.py
Generates and evaluates various types of weak classifiers, computing intergral images, and training AdaBoost for boosting-based face detection.
loaded trained model
Decision
Loaded testing images
Boosted Predict
neither
2
1
Terminator
face_detection_model.pkl
The final trained model using cascade classifiers and AdaBoost.
t2py
Advanced testing of trained model
testing.py
Includes functions to test whether an image contains a face or not, using cascades and evaluation metrics like precision and recall.
t1.py
Basic testing of trained model
newSkin.py
Advanced skin detection method that outputs a mask where only detected skin is shown
2
Decision
1
Basic Outputs
Where all basic testing outputs are located
nms.py
Filters overlapping boxes by position and scores.
OutputAdvanced
Where all of the advanced test outputs are located

Fig. 4.1 Flow chart of basic and advanced testing functionality

Following our training our testing functionality is next. It is called with test.py following with a 1, for a basic test, or 2, for an advanced test, given by user input. While both tests use data from testing.py, the basic testing when called it gets the loaded images/data from processing.py along with the trained model from model.py, after which it conducts the simple testing methods outputting to the directory “important_outputs/outputBasic”.

Following is the advanced testing which uses the boosted predict function from boosting.py the loaded trained model from train.py. During the testing the advanced method calls the newSkin.py, our advanced skin detection, to create a mask to limit the amount of data needing to be tested. After the test is finished, it is then put through our nms.py functionality to refine the detection boxes outputted onto the file, where it’s finally saved under the directory “important_outputs/outputAdvanced”.

V. Visualization

The data below comprises different testing cases, as can be seen the advanced skin detection mask led to more accurate results and that is due to a more complex skin detection and Non-maximum Suppression (NMS) functionality.

A. Testing Data

Shown below are the original images, along with the corresponding outputs for each basic and advanced skin detection masks, and tests. Though the results shown in the console are similar in both cases, as can be seen below, the advanced test is theoretically better. If given a larger dataset the advanced skin detection will outperform the basic skin detection, as the skin detection and NMS are more complex in the advanced skin detection, causing more accurate masks to be produced. "Soft-NMS leads to noticeable improvements in average precision measured over multiple overlap thresholds for state-of-the-object detectors on standard datasets,"[3] Our implementation of the NMS function was highly efficient, but it resulted in bounding boxes that were smaller than ideal. To remedy this, we fine-tuned the process to encourage the creation of appropriately sized bounding boxes. Moreover, we decided to include it in not only the advanced skin detection but also our own NMS function.

TABLE II

Testing Basic V.S Advanced skin detection with Face Photos, the data set contains both faces and non-faces

Type of detection	**Data**			
	True Positives	**False Positives**	**Precision**	**Recall**
Basic	49	12	.80	**1.00**
Advanced	49	17	.74	**1.00**

TABLE III

Testing Basic V.S Advanced skin detection with Cropped Faces, the dataset contains only faces

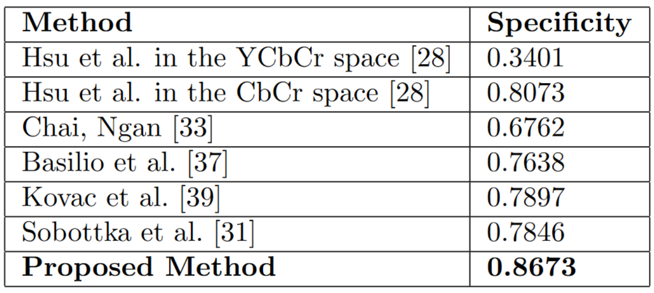
Type of detection	**Data**			
	True Positives	**False Negatives**	**Precision**	**Recall**
Basic	581	189	.75	**1.00**
Advanced	581	189	.75	**1.00**

TABLE IV

Testing Basic V.S Advanced skin detection with Non Faces, the dataset contains only non-faces

Type of detection	**Data**			
	True Negatives	**False Positives**	**Precision**	**Recall**
Basic	34	2	.94	**1.00**
Advanced	34	2	.94	**1.00**

The data in the tables II, III, IV reveal that while both basic and advanced skin detection tests perform with high accuracy, the advanced test demonstrates a clear advantage in identifying true skin detections. This advantage is anticipated to grow with larger data sets, as suggested by the consistent true positive rates and perfect recall across the tests. The advanced testing method, despite a slight increase in false positives, is expected to refine its precision with more extensive data, as evidenced by Brancati et al in figure 4.2. Furthermore, when used in conjunction with Soft-NMS, the long scale accuracy is more optimal.



Method	**Specificity**
Hsu et al. in the YCbCr space [28]	0.3401
Hsu et al. in the CbCr space [28]	0.8073
Chai, Ngan [33]	0.6762
Basilio et al. [37]	0.7638
Kovac et al. [39]	0.7897
Sobottka et al. [31]	0.7846
Proposed Method	**0.8673**

Fig. 4.2 Specificity for a set of non-skin images.

B. Image Outputs

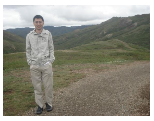


Fig. 5.1 Original image DSC04545

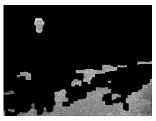


Fig. 5.2 Basic skin mask of Original image DSC04545

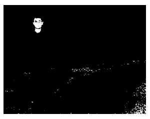


Fig. 5.3 advanced skin mask of Original image DSC04545

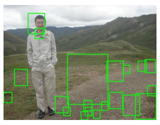


Fig. 5.4 Basic test of Original image DSC04545

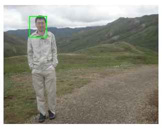


Fig. 5.5 Advanced test of Original image DSC04545

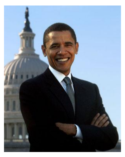


Fig. 6.1 Original image obama8

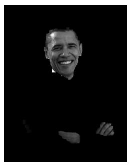


Fig. 6.2 Basic skin mask of Original image obama8

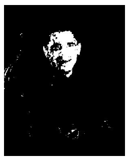


Fig. 6.3 advanced skin mask of Original image obama8

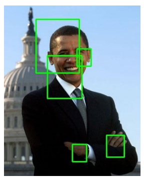


Fig. 6.4 Basic test of Original image obama8

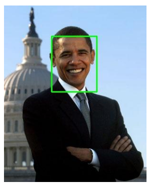


Fig. 6.5 Advanced test of Original image obama8

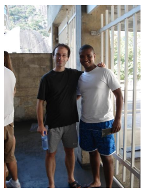


Fig. 7.1 Original image DSC01181

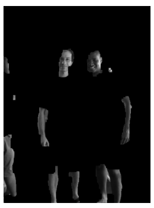


Fig. 7.2 Basic skin mask of Original image DSC01181

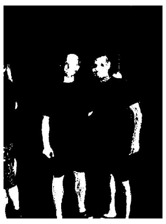


Fig. 7.3 Advanced skin mask of Original image DSC01181

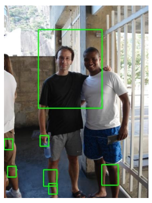


Fig. 7.4 Basic test of Original image DSC01181

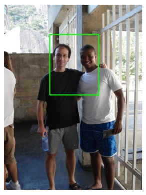


Fig. 7.5 Advanced test of Original image DSC01181

VI. Conclusion

In conclusion, our project VIZN stands out in facial detection by seamlessly blending AdaBoost, Skin Detection, and Cascade Classifiers for a highly accurate face detection model. AdaBoost forms the core, Skin Detection fine-tunes the focus, and Cascade Classifiers speed up the process by ignoring non-facial areas. Our program delivers a simple interface to advanced functionalities.

References

[1] "UCI Machine Learning Repository," archive.ics.uci.edu. https://archive.ics.uci.edu/dataset/229/skin+segmentation (accessed Dec. 07, 2023).

[2] N. Brancati, G. De Pietro, M. Frucci, and L. Gallo, "Human skin detection through correlation rules between the YCb and YCr subspaces based on dynamic color clustering," Computer Vision and Image Understanding, vol. 155, pp. 33–42, Feb. 2017, doi: https://doi.org/10.1016/j.cviu.2016.12.001.

[3] N. Bodla, B. Singh, R. Chellappa, and L. S. Davis, "Soft-NMS -- Improving Object Detection With One Line of Code," arXiv:1704.04503 [cs], Aug. 2017, Available: https://arxiv.org/abs/1704.04503